A Convolutional Neural Network Approach for Detecting NAO Robots

Individual Report

B.Sc. Yixi Zhao
B.Sc. Zhen Zhen
B.Sc. Bowen Ma
B.Sc Siyu Chen
Supervisors: M.Sc. Quentin Leboutet, Prof. Dr Gordon Cheng

***Abstract*—The detection of opponents is essential for developing an effective strategy against the competition during the match. In this paper, we present an approach using a convolutional neural network for robot detection. Since robots will have different poses during the match, i.e., running, kicking, laying on the ground, the application of neural network, which has generalization capabilities, is a research worthy approach. We first preprocess the camera image by extracting parts of the image, which might contain one or more robots, using a bounding box. These extracted images are then fed to a convolutional neural network, that decides whether the image contains an actual robot by computing a probability. If a robot has been identified, it's jersey color will be used to determine whether it is an opponent or not. The position of the opponent robot will then be converted to the field coordinates.**

***Index Terms*—Convolutional Neural Networks, Binary Classification, RoboCup, NAO, Computer Vision, Robot Detection, Computational Efficiency**

I. Introduction

The key to winning in a football game is scoring a goal. This process requires the cooperation of the entire teammates, and adjust the strategy according to changes in the opponent. Detecting opponents is therefore significant for winning the entire game. But the current Hulks system lacks the step to identify and detect the opponent.

Unlike ball detection, it is difficult to find universal features, since the robot has many different postures (standing, lying down, etc.) during the game. To solve this issue, we intend to implement an opponent detection using a convolutional neural network (CNN), since a well-trained network can generalize very well. Another motivation is because a neural network approach has never been used in the HULK's framework to detect opponents, we want to verify its feasibility through this design. To this end, our group proposed several different CNN-based solutions. By modifying the model parameters, we thus obtain different CNN models for training.

A. Related Works

In [1], the B-Human team pre-processes in the upper image with 2% min-max normalization. Then they divide the lower Image into small regions and detect the change of contrast in the horizontal, the vertical, or both of these directions. They use a neural network to detect the robots. The information will be propagated from the upper to lower image. This method could be helpful to our work, but our robot is of a different version than that of the B-Human team, so the memory may be limited.

In [2], the HTWK team first gets the hypotheses generation, which excludes regions that are above the field limitations or with low contrast characteristics. Then the classification is performed by generating hypotheses with a deep convolutional neural network (CNN). They use Caffe as a deep-learning framework and optimize it for the NAO. They improve the robot detection by sharing the CNN with the ball and penalty spot detection. Although this method achieves high accuracy, the CNN framework Caffe is not suitable for our robot, because it can't be used in the Hulks with OpenCV. Additionally, real-time can't be achieved.

In [3], two different CNN based NAO robot detectors, that can run in real-time while playing soccer, are proposed. One of the detectors is based on the XNOR-Net and the other on the SqueezeNet. Each detector can process a robot object-proposal in 1ms, with an average number of 1.5 proposals per frame obtained by the upper camera of the NAO. But they use darknet as the framework, it also can't be connected to the Hulks with Open-CV.

The aim of [4] was to develop a multi-class object detector to perceive balls and robots at the same time on the NAO robotic system. They implement a framework for collecting and labeling data, training a CNN for classification, comparing the optimization approaches of MCTS and GAs with and without elitism. Finally, a framework to inference the resulting network on the NAO was developed and evaluated. But it requires more computation power with a quad-core CPU with an integrated GPU.

II. Theoretical Overview

Our approach consists mainly of the following steps, namely, preprocess the camera image and obtaining a cut-out image of robot candidates, deciding whether a candidate is a robot using convolutional neural networks, classifying the robot as an opponent, and adding its position to the field

Institute for Cognitive Systems (ICS), Technical University of Munich, Faculty of Electrical Engineering and Computer Science, 45/II Karlstraße 80333 Munich, Germany.

coordinates. The hyperparameters used are discussed in the section technical details.

A. Image Preprocessing

The incoming camera image requires preprocessing before being inputted into the convolutional neural network. This step is needed to reduce the computational cost of feeding an image through a network. The goal of this step is to scan the camera image for possible robot candidates and create a bounding box around them. Each image object with a bounding box is then extracted from the original camera image and will be used for further processing. This way the convolutional neural network only has to process the image object and decide whether its a robot or not rather than computing the entire input image.

For the preprocessing, we first create a filtered image out of the original camera image. Then we create a sliding window, which slides through the filtered image and keeps those windows, which contain an area of interest. The overlapping sliding windows are then combined to create a candidate bounding box.

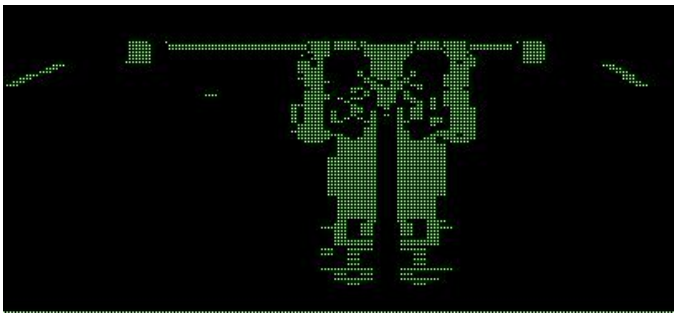

Fig. 1: The filtered image created from a simulation. The robot, the field lines, and the goalposts are set as one, while the background is set as zero.

1) Filter The Camera Image: Given the original camera image of type Image422, we create a two-dimensional matrix of the same size and set it to zero. The pixel values of this matrix are set to one using the horizontal scanlines from the already implemented module. Specifically, the horizontal scanlines also contain information about detected rising and falling edges, i.e., the edge between field to object and object to field. If a scanline contains both the rising edge (edge_{start}) and its corresponding falling edge (edge_{end}), if the difference between these edges is smaller than a threshold ($theta_{hscanline}$), and if the image pixel is within the field border, then every pixel between these will be set as one (Algorithm 1). The resulting filtered image is shown in Fig. 1

2) Obtain Area of Interest: After obtaining the filtered image, the difference between a robot candidate and the background becomes obvious, as image areas containing a candidate will be mostly ones, while the background is mostly zero. To find the areas of interest, we decided to use a sliding window, which shifts over the entire filtered image with certain step size and computes the percentage of the area of interest (P_{AoI}), i.e., percentage of how many areas are being covered by a possible candidate using the equation

Algorithm 1: Create filter image

```
Data: Image422
Result: ScanImage
initialize 2D ScanImage;
while until last horizontal scanline do
    if edge_end − edge_start < θ_hscanline then
        while from edge_start until edge_end do
            in field pixel becomes one;
    else
        pixel remain zero;
```

$$P_{AoI} = \sum pixel/(height_{window} * width_{window}). \quad (1)$$

Equation 1 summarizes every pixel within the sliding window and divides it through the same window with every pixel being a one. If P_{AoI} is above a threshold (θ_{AoI}), the corresponding sliding window is kept for further processing.

The sliding window starts with the upper left corner at the image coordinate of (0, 0). After each iteration, we shift the window in the x-axis by a predefined step size, and once the window reaches the image width, the sliding window is moved to the start of the next line with the coordinate (previous y value + step size, 0). These steps are repeated until the lower bound of the sliding window reaches the image bottom (Algotihm 2). The image 2 illustrates the result of this step.

Algorithm 2: Create filter image

```
Data: ScanImage
Result: Area of Interests
while until end of image height do
    while unitil end of image width do
        compute P_AoI;
        if P_AoI < θ_AoI then
            save sliding window;
        else
            dismiss current window;
        shift window by step size;
```

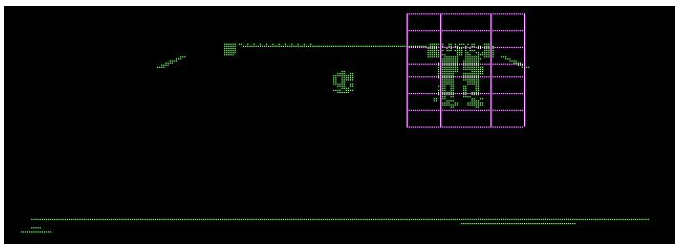

Fig. 2: The area of interest detected using the sliding window approach. The robot in the far-field has been detected in comparison to the goal post, the ball, and field lines.

3) Create Bounding Box of Robot Candidates: After finding the areas of interest, which are covered by many smaller windows, the next logical step is to combine them to obtain a bounding box for the area. This is achieved by checking whether two bounding boxes overlap or not. The overlapping boxes are then combined to make a new bigger box. An example is shown in Fig. 3. The coordinates that define these bounding boxes are saved for further processing.

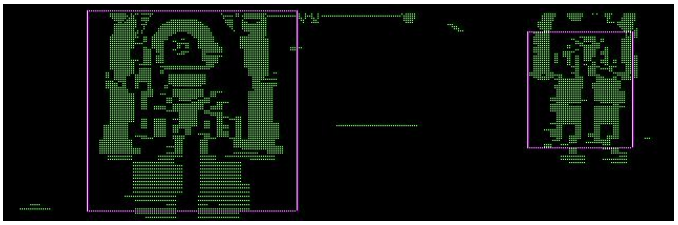

Fig. 3: Bounding boxes for robot candidates have been created from valid sliding windows.

B. Convolutional Neural Network

Due to the performance limitations of NAO V5, we must design a simple neural network. Otherwise, the robot needs too much time processing one single image. If real-time processing cannot be achieved, the robots will be to slow to react. Unlike ball detection which detects only fixed features, the robots can have different poses. This means more parameters are required to detect these features. With these criteria in mind, we designed a CNN for binary classification, instead of a CNN that draws a bounding box. Our CNN will compute a probability for the robot candidate and make a classification decision based on that.

1) Input for CNN: Instead of the entire camera image, only the bounding boxes with robot candidate are cut out and fed to the CNN. The cut-out image is transformed into a gray-scale image (Fig 4) and resized to an image of 30x30. Depending on the CNN model and the images used for training, the cut-out bounding box has to be normalized using mean-centering, re-scaled such that the values lie within zero and one, or both. However, normalization is not strictly needed, since the pixel values are limited between [0, 255]. The effect of normalization is discussed in the chapter result.

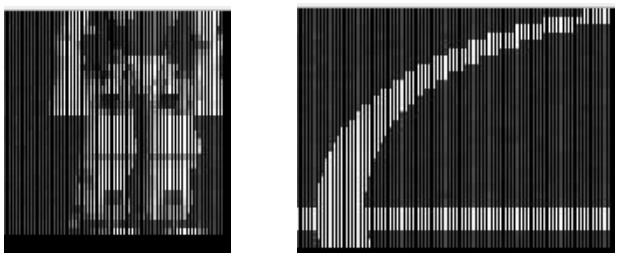

Fig. 4: Gray-scaled images of robot candidates (left panel: upper camera, right panel: lower camera) obtained by running the simulation. These images will be classified by the CNN one by one while running the simulation.

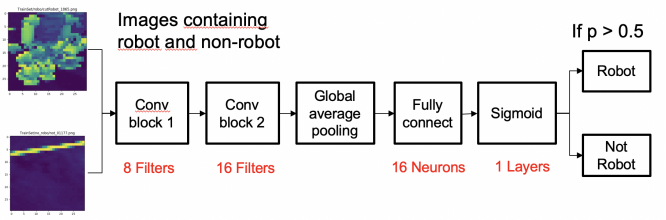

Fig. 5: One architecture of convolutional neural network

2) CNN Architecture: Each convolutional block includes Conv2D, Activation Relu, Batch Normalization, Maxpooling2D, Dropout. Convolution layers are used to extract features. The role of the activation relu is to increase the nonlinear relationship between the layers of the neural network. If we go through Batch Normalization, we can make the output of each layer mean 0 and standard deviation 1, that is, the data is concentrated near the y-axis. This makes the gradient larger and solves the problem of gradient disappearance. The main role of max-pooling is to retain the main features while reducing parameters (dimensional reduction, an effect similar to PCA) and calculation volume, preventing overfitting and improving the generalization ability of the model. In each training batch, dropout can significantly reduce overfitting by ignoring a part of the feature detectors (the hidden node value is 0). The output is computed by a sigmoid layer, which returns a probability. Above a predefined threshold of 0.6, the robot candidate is validated as an actual robot (Fig. 5).

C. Loss, Metrics, and Optimization

We decided to use the binary cross-entropy as the loss function since we aim to perform binary classification. Through the usage of this loss function, miss-classification will be penalized. For the metrics, we decided to use the accuracy to check the performance of our CNN classifier. The optimizer used is the adam optimizer with a learning rate of 0.0001 and a decay of 0.001.

D. Opponent Identification

Although the NAO robots look the same, each team has a unique jersey color. Therefore, one way to differentiate a teammate from an opponent is by recognizing the jersey color of our team. A robot wearing a jersey with a different color is then classified as an opponent.

1) Convert YCrCb to HSV Color Space: YCrCb is a scaled and offset version of the YUV color space. The camera images of the NAO robot uses YCrCb color space instead of YUV color space. Compared with YUV, YCrCb color space is applied in digital video. Furthermore, the YCrCb color space is less affected by the brightness of the lighting condition. The pixel values have therefore fewer fluctuations. Y refers to the luminance component, Cb refers to the blue chrominance component, and Cr refers to the red chrominance component.

However, YCrCb is only sensitive to the blue and red chrominance components, meaning that, while two different colors containing red and blue can be differentiated, other

colors such as yellow will be likely miss-classified. To be able to differentiate any given two colors, we transform YCrCb into HSV color space. Although HSV space is susceptible to environmental brightness, it is sensitive to all of the colors.

HSV is a representation of the points in the RGB color model in a cylindrical coordinate system, which is more intuitive than the geometric structure RGB based on the Cartesian coordinate system.

HSV is Hue (H), Saturation (S), and Value (V). Hue (H) is the basic attribute of color, which is usually the name of the color, such as red, yellow, etc. Saturation (S) refers to the purity of the color. If the value S is higher, the color is more pure. As the value S decreases, the color becomes more gray. Value(V) represents the brightness of the color, ranging from 0 to 1.

2) Color Recognition: After identifying the robot using the CNN, we only have to read the pixel values within the bounding box to search for the jersey color. After the YCrCb to HSV color space conversion, we perform histogram equalization processing. This process ensures that the histogram of the image covers all possible gray levels and is evenly distributed so that the resulting image has high contrast and variable gray tones. Then the image is binarized and the color of our team's jersey is set to white, while everything else becomes black.

The noise in the image from the obtained image is removed and some domains are connected, so the target area becomes a whole part. The operations can only identify nearby opponents and ignore distant robots or target color areas that are not robots. By reading the pixel matrix of the image, the value of the recognized color is set to 255, otherwise 0. Each pixel point of the image matrix is read separately. If the points are not 0, they are accumulated. Finally, the number of white pixels can be calculated, which is the number of target color points. This step is also used to remove noise and identify large target color areas of nearby robots.

If the number is larger than the threshold number, the jersey color is identified as that of our team (See algorithm 3).

Algorithm 3: Color Recognition

Data: CutImage
Result: This is opponent or teammate.

```
Transfer YCrCb image into RGB image;
Transfer RGB image into HSV image;
HSVSplit;
iLowH, S, V←Low Values of H, S, V;
iHighH, S, V←High Values of H, S, V;
Histogram equalization;
calculate the number n of white points;
if n ≥ threthhold value then
    This is an opponent.;
else
    This is an teammate.;
```

E. Opponent Localization

To visualize the detected robots in MATE by using the obstacleMap template in real-time, it's rather important to estimate the positions of the detected robots. Through the process of neural network introduced above, we can get a series of relatively accurate detected robots. However currently what we get are just two-dimensional positions of the detected robots in the pixel coordinate system, which is not enough to be visualized in MATE. What we need to know is the pose of each detected robots in the field coordinate system. As a result in our work what we did is the transformations of the robot's position in the field of view from pixel coordinate system to camera coordinate system, then to robot coordinate system and at last to field coordinate system which we are expected. For the other task of position estimation, we would like to fuse the positions of the robot which is detected by different robots. In this situation, we have to take into account how sure a robot is, about its position on the field, which is also the accuracy of the self-localization. We use the standardized mean error of evaluation of the self-localization from different robots to assign different weights to these robots, to optimize the consequence of the position-estimation.

1) Position Estimation without Data Fusion: In our work, we decided to use the middle points in the bottom-line of bounding-boxes as the needed robot positions in the pixel coordinate system. That's because in general these points are actually contacted points between the robot's feet and the football field if the entire body of the robot is detected. This can also benefit to ignore the effect of the z-axis in the field coordinate system. Equation 2 shows the process of the transformation.

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{internal} \underbrace{\begin{bmatrix} R & T \\ \overrightarrow{0} & 1 \end{bmatrix}}_{external} \begin{bmatrix} X_f \\ Y_f \\ Z_f \\ 1 \end{bmatrix} \tag{2}$$

In the equation we can see, the matrix of f_x, f_y, u_0, v_0 is actually the internal parameters of the camera, which can be obtained from the camera-matrix data. The RT(Rotation and Transformation) matrix is the external parameters that can be obtained based on the robot's self-localization.

2) Position Estimation with Data Fusion: According to [5] we know that for HULKs the latest self-localization technology is based on Unscented Kalman-Filter (UKF). In our work, we take account of the "MeanEvalError" generated during the self-localization process. We normalize the inverse of this "MeanEvalError", whose values are distributed between 0 and 1. Finally, we define it as the weight-values, which are given to the different calculated robots' pose in-field coordinate system detected by the different robots. Equation 3 shows how we fuse the position estimates of each robot.

$$\underbrace{\left[\frac{1}{\text{MeanEvalError1}}\right]}_{normalized} * \text{POS1} + \underbrace{\left[\frac{1}{\text{MeanEvalError2}}\right]}_{normlized} * \text{POS2} \tag{3}$$

III. Technical Details

We build and trained our CNN model on Python using Tensorflow Keras. The CNN model is created as an object from a class. The main script calls the object builds, trains, tests, and saves the model. However, since this model is saved as an h5 file, it can't be recognized by OpenCV. Our solution was to transform this h5 file into a readable pb file. This is done by converting the h5 file to a frozen pb file first and then convert it again to an inference pb file, which can be interpreted by OpenCV. With the exception of the Flatten layer, all the other layers can be recreated when compiling the C++ code of the HULKs framework. This problem was overcome by replacing it with a global average pooling layer, which did not interfere with the result but rather reduced the number of parameters greatly.

Aside from the CNN model, everything else has been integrated directly into the HULK's framework, specifically the RobotDetection module.

A. Hyperparameters of the Preprocessing

The value of $theta_{hscanline}$ is obtained trough testing in the simulation and the real world. If the value is too large, background noise will be included, on the other hand, if the value is too small, a robot will not be detected if it stands to close to the camera.

The window size and step size for the upper and lower camera differs, since robots in the upper camera are usually in the far- or mid-field and therefore quite small compared to the robots appearing in the lower camera, which are very close. The optimal window size and step size are obtained through testing in the simulation and in the real world.

B. Training CNN

We use the dataset from [6]. We try to obtain training images using different data augmentation methods, these include cropping, rotating. Figure 6 provides an example of input training images. All images must be resized to 30*30. The end result is: 2016 non-robot pictures, 2233 robot pictures.

Fig. 6: Training images sorted into no-robot class (left panel) and robot class (right panel)

C. Hyperparameters of CNN

All hyperparameters of CNN are shown as table I below. CNN layers represent the list of convolutional filters. FC layers represent the list of fully connected layers, the output should be 1, i.e. [64, 1]. We tested CNN with different hyperparameter configurations. The training loss and accuracy, the validation loss and accuracy, as well as the test accuracy are obtained during training and testing with the online dataset. After testing, we get the best filter size: [8, 16] [16, 1].

TABLE I: The hyper-parameter configurations and the corresponding training, validation, and testing results

	Hyperparameters			
CNN layers	[4, 8]	[8, 16]	[8, 16, 32]	[16, 32, 64]
FC layers	[8, 1]	[16, 1]	[32, 1]	[64, 1]
Trainable Par	441	1585	7089	27745
Total Par	465	1633	7201	27969
train loss	0.1858	0.0801	0.0341	0.0053
train acc	0.9422	0.9736	0.9896	0.9997
val loss	0.1950	0.0797	0.0441	0.0252
val acc	0.9430	0.9788	0.985	0.9886
test acc	0.945	0.9702	0.979	0,979

TABLE II: The values of color

Color	**High Value of H**	**Low Value of H**
Orange	22	0
Yellow	38	22
Green	75	38
Blue	130	75
Violet	160	130
Red	179	160

D. Opponent Identification

We use the functions under OpenCV to convert the YCrCb model to the RGB model and then to the HSV model. In OpenCV, H $\in$ [0, 180], S $\in$ [0, 255], V $\in$ [0, 255]. The H component expresses the color of the robot. The values of S and V must be within a certain range because S represents the degree of mixture of the target color and white, V represents the degree of mixture of the target color and black. Through experiments, S and V values are both between 90 and 255. The H value of some basic colors can be set as table II.

E. Opponent Localization

In our work we use the bool function "pixelToRobot" which is provided in the "CameraMatrix.hpp" file. What we used as the input positions are the middle point in the bottom-line of the bounding boxes. As a consequence, we got a series of poses of the detected robots in the field coordinate system. In the "ObstacleFilter" file, there exists a function which can help to update the detected robots' positions according to the self-localization in real-time. As a result when we push back the value of positions to the "RobotPosition" data-file, the positions of the detected robots in field of view could be shown as pink circles in MATE.

IV. Experiments

A. Description of the Hardware and Control Interface

We performed simulations using SimRobot package on a computer with 8 Intel(R) Core(TM) i7-6700HQ CPU processor and a RAM of 7572044 kB. The operating system is Ubuntu 16.04.

B. Description of the Experiments

1) CNN Training Result: The training result is shown in Fig. 7. It shows that our model indeed learns the feature and improves the accuracy.

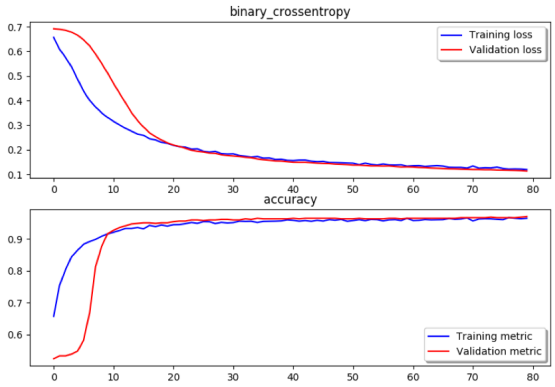

Fig. 7: The typical error plot during training of the convolutional neural network

By testing various hyperparameter configurations, we obtained training and testing result as shown in table I. From this table, we can see that too many layers and filters do not automatically improve the result, but rather tends to overfit since the test accuracy is lower than during training. Additionally, the test result did not improve all that much.

On the other hand, too few layers and filters will lower accuracy. This is expected since fewer trainable parameters result in fewer features being learned. The best hyper-parameter configuration, that we have tested, consists of 2 CNN layers with 8 and 16 filters, followed by a fully-connected layer with 16 neurons.

C. Testing using the Simulation

During the testing by using the simulation, the robot could detect multiple opponents in various ranges at once. Additionally, once an opponent has been detected, the bounding box remained, even when the distance increased or decreased. If the opponent is too close, i.e., fills up almost the entire upper camera and most of the lower camera, then the edge detection used in the preprocessing becomes unreliable, resulting in no detection or detection of body parts like the shoulder or arm. Most of the robot candidates can be correctly classified as the figure 8 shows. However, occasionally, a no-robot candidate will be miss-classified as a robot, and if two robots are standing very close to another, they will be detected as one robot. The average time consumption of our module while running the simulation took 0.001 seconds, with the worst case being 0.016 seconds. Additionally, the detected robots could be perfectly visualized as pink circles in MATE, just like figure 9.

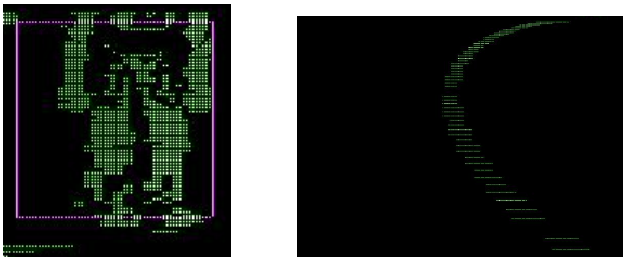

Fig. 8: Given the images from Figure 4 as input while running the simulation, the CNN has correctly identified the robot candidate on the left panel as a valid robot while correctly dismissing the candidate on the right panel as no-robot.

Fig. 9: Visualize the detected robots in simulation with pink circles.

D. Testing in the Real World

Testing in the real world with the NAO V5 robots, the result were less optimal compared to the simulation. The biggest issue was due to the unstable light condition. Since our preprocessing uses edge detection computed by another module, it does not always detect the contrast lines. Additionally, the robot shakes while walking, which also causes the vision to be shaking as well.

Another issue is the amount of time and computation required by running the CNN. The average time to process one camera image took 0.016 seconds, while the average CNN processing took 0.01 seconds. In the worst case, it can take up to 0.056 seconds. Furthermore, 60 percent of the NAO's computation power was used for processing this module, which means, the robot won't have capacity to run any additional functions.

V. Conclusion

In conclusion, because of the limitations of the Nao v5 version in this project, we tested multiple simple neural network models in our work. By using the existed dataset on the internet and the dataset we collect from the robot's camera in the real scenario, we trained and improved the neural network. Regarding the training results, we achieved a very high accuracy of almost 98 percent. During the real field tests, the robot manages to detect the opponent robots even under the influence of unstable light conditions. However, some problems remain, such as the time consumption of image processing and dealing with the situation of multiple robots. For future work, if CNN is still to be used on the current version of the NAO robot, then the preprocessing should be improved, while the CNN should be made even smaller.

References

[1] U. Bremen, "Detecting other robots," in B-Human Team Report and Code Release, pp. 71–75, 2019.
[2] N.-T. HTWK, "Black and white ball detection," in HTWK Team Research Report, pp. 10–12, 2019.
[3] K. L.-T. Nicolás Cruz and J. R. del Solar, "Case study: Real-time nao detection while playing soccer," in Using Convolutional Neural Networks in Robots with Limited Computational Resources: Detecting NAO Robots while Playing Soccer, pp. 4–10, 2017.
[4] G. C. Felbinger, "Convolutionalneuralnetworks," in Optimal CNN Hyperparameters for Object Detection on NAO Robots, pp. 8–11,27–30, 2018.
[5] P. G. C. K. Y. K. R. K. P. L. K. N. L. P. N. R. T. S. M. S. E. S. F. W. F. W. Darshana Adikari, Georg Felbinger, "Hulks team research report 2018." https://hulks.de/_files/TRR_2018.pdf.
[6] M. Szemenyei, "Robust real-time object detection for the nao robots." https://github.com/szemenyeim/ROBO.